90528 CGAR 2021/22

Computer Graphics - Homework 2

Basic Raytracing Effects

The goal of this assignment is to implement basic ray tracing effects such as shadows, reflection, refraction, and depth of field.

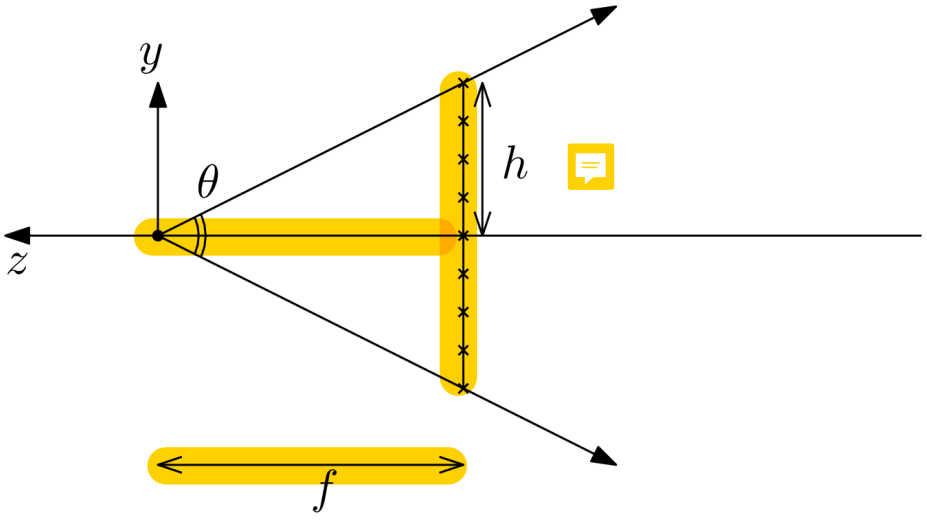
Using Eigen

In all exercises you will need to do operations with vectors and matrices. To simplify the code, you will use Eigen. Have a look at the Getting Started page of Eigen as well as the Quick Reference page for a reference of the basic matrix operations supported.

Preparing the Environment and Submission

Follow the instructions on the *General instructions* document to set up what you need for the assignment.

Ex.1: Field of View and Perspective Camera



The field of view of a perspective camera represents the angle formed between the center of the camera and the sensor (aka the pixel grid through which rays are shot). The focal length is the distance between the camera center and the sensor, and is called `f` in the figure above.

1. Fill the starter code to compute the correct value of `h` (`scale_y` in the code).

2. Implement the perspective camera similarly to Assignment 1.

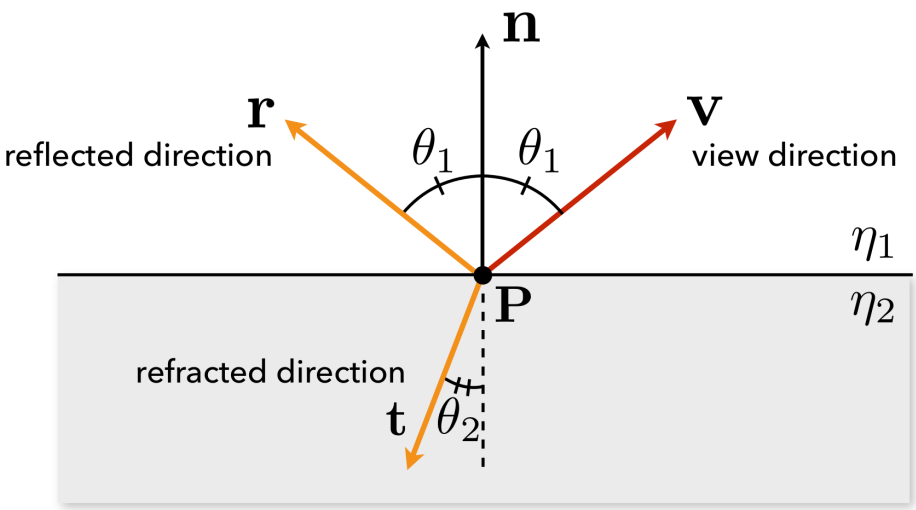
Ex.2: Shadow Rays

To determine if a point is in the shadow of another or not, one must cast a ray from this point to the different light sources in the scene. If another object lies in between the point and the light source, then this light does not contribute to the color of the point.

Tasks

1. Fill the starter code to implement shadow rays.
2. Show what happens when the shadow ray is not offset by a small epsilon value.

Ex.3: Reflection & Refraction



$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

(Figure: Wojciech Jarosz)

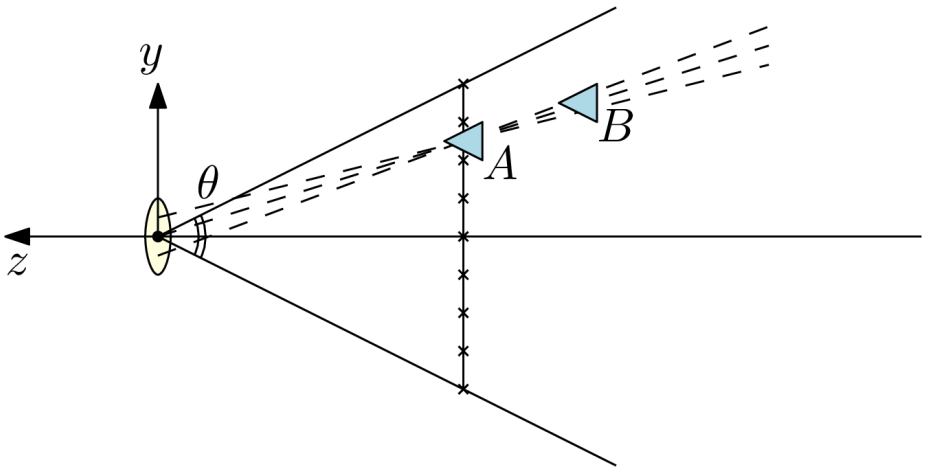
To render mirrors and transparent materials, shading must also consider objects that could be reflected or seen through the surface hit by the camera (primary) rays. This can be achieved by shooting new rays from the hit position to the scene with a new direction.

The direction of the reflected has been given in class, and can be expressed as `r = 2n(n·v)-v`. The direction of the refracted ray can be derived from the Snell-Descartes law indicated on the figure.

Tasks

1. Derive the formula for the direction of the refracted ray based on the Snell-Descartes law.
2. Fill the starter code to implement reflected and refracted rays. Don't forget to decrease the counter to limit the maximum number of 'bounce' a ray can make, and be sure to check for total internal reflection when implementing refraction (it may happens that there is no refracted ray if the incident direction is too low).

Ex.4: Depth of Field



A basic depth of field effect can be achieved in our simplified camera model. Instead of only one ray per pixel, one can shoot several rays per pixel as illustrated by the figure above. In practice, this means the hole of our camera no longer has an infinitesimal size, a parameter known as the aperture in real-world cameras.

Here, the focal plane of our camera corresponds to our pixel grid (which we have put at a distance `f` from the camera center). This means that object A, which is close to the focal plane, will appear sharper than object B, which is more distant.

To implement a depth of field effect, you will need average the contribution of several rays per pixels, sampled randomly as suggested in the picture.

Tasks

1. Fill the starter code to implement depth of field.
2. Modify the input scene to move the rightmost sphere more into focus. Experiment with different settings of this effect.

Ex.5: Animation

Generate an animation by moving or changing size/properties of the object in the scene and exporting a series of images. You can export a sequence of pngs and look at them in sequence, or directly export an animated gif file by including `#include <gif.h>` and using the following code snipped:

```
const char* fileName = "out.gif";
vector<uint8_t> image;
int delay = 25; % Milliseconds to wait between frames
GifWriter g;
GifBegin(&g, fileName, R.rows(), R.cols(), delay);

for (unsigned i=0;i<10;i++)
{
    // Generate R G B A matrices

    write_matrix_to_uint8(R, G, B, A, image);
    GifWriteFrame(&g, image.data(), R.rows(), R.cols(), delay);
}
GifEnd(&g);
```

Starting Code

After compiling the code following the process described in the general instructions, you can launch the program from command-line as follows:

```
mkdir build; cd build; cmake ..; make
./assignment3 ../data/scene.json
```

Once you complete the assignment, you should see result a picture generated in your folder.

After implementing perspective camera, specular highlights, shadows, reflections, refractions and depth of field, the result of your code should like this: